1. Json0200: The What and Why of JSON
2. Json0200R: Review
3. Json0205: Getting Started
4. Json0205R: Review
5. Json0210: Structure of the json-simple Java Library
6. Json0210R: Review
7. Json0215: Encoding JSON Strings
8. Json0215R: Review
9. Json0220: Decoding JSON Strings
10. Json0220R: Review
11. Json0225: Encoding JSON Arrays
12. Json0225R: Review
13. Json0230: Decoding JSON Arrays
14. Json0230R: Review

Json0200: The What and Why of JSON
Learn what JSON is and why, as a Java programmer, you should care about JSON.

Revised: Sun Jul 03 10:16:32 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of contents

Preface

What is JSON?

The online document titled Introducing JSON begins as follows:

> **Note:**
> "JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language."

Similarly, the online document titled Java API for JSON Processing: An Introduction to JSON begins as follows:

> **Note:**
> JSON (JavaScript Object Notation) is a lightweight, text-based, language-independent data exchange format that is easy for humans and machines to read and write. JSON can represent two structured types: objects and arrays. An object is an unordered collection of zero or more name/value pairs. An array is an ordered sequence of zero or more values. The values can be strings, numbers, booleans, null, and these two structured types.

It is important to note that even though JSON is based on JavaScript syntax, JASON is not JavaScript nor is it any other programming language. In fact, it is not a programming language at all. As stated above, JSON is simply *"a lightweight, text-based, language-independent data exchange format"* -- nothing more and nothing less.

Figure 1 shows a JSON text string containing name/value pairs as well as nested arrays.

Note: Figure 1. Example JSON text.

```
{
  "game":
    [
      {"cards":
        [
          "2-club","3-heart","4-diamond","5-spade"
        ],
        "name":"Tom"
      },
      {"cards":
        [
          "4-heart","5-heart","6-club","7-diamond"
        ],
        "name":"Joe"
      }
    ]
}
```

(This JSON text will be used in a Java program in a future module.)

Why should you care about JSON?

This book is being written and published under the following assumptions:

- You are interested in Java programming.
- You are interested in web development involving Java programming.

- At some point in the future, you may become interested in Big Data .
- At some point in the future, you may become interested in NoSQL databases such as MongoDB and Couchbase .

As stated in the InfoWorld article of August 25, 2014:

Note:
"Web developers love JSON (JavaScript Object Notation). Like XML, it's a human-readable format for transmitting data, except it's a whole lot easier to work with than XML. ... Several NoSQL databases -- including the wildly successful MongoDB and Couchbase -- store data in JSON documents natively."

According to How JSON and Big Data Will Shape the Internet of Things , the author writes:

Note:
"To answer the question of why JSON would be the most widely used format for the Internet of Things, one only need look at the rapid development of Raspberry Pi, which started a little over two-and-a-half years ago, and has gained massive traction worldwide. This credit-card sized microcomputer is extensible, and a recent project called RaZberry has turned it into a device capable of controlling your home automation through - you guessed it - a JSON interface. With future development of the Internet of Things, the proliferation of JSON as the preferred data delivery mechanism will only increase.
Even more interesting is how this data can be fed into a Big Data cluster to perform predictive modeling and analytics. Just over a year ago, Google BigQuery added support for JSON and explicitly mentions how sensor data and its attributes can be measured as a consequence. With time, it is only inevitable that developers in other Big Data ecosystems will use JSON

when setting up their clusters to perform analytics from the various source data from the Internet of Things."

According to the aforementioned Google BigQuery support for JSON article:

Note:
"JSON is the data format of the web. JSON is used to power most modern websites, is a native format for many NoSQL databases hosting top web applications, and provides the primary data format in many REST APIs. Google BigQuery, our cloud service for ad-hoc analytics on big data, has now added support for JSON and the nested/repeated structure inherent in the data format."

I could go on providing similar quotations, but the bottom line is that if you anticipate your Java programming career taking you into the world of Big Data or into The Internet of Things at some point in the future, you probably need to learn how to write Java programs that parse, generate, transform, and query JSON.

Not a book about JSON

This book is not intended to teach you about JSON. There are numerous tutorials on the web that you can access for that purpose *(see Online references)* . This book is intended to teach you how to use the json-simple Java library, *(which is one of several available libraries),* to parse, generate, transform, and query JSON.

The page at http://www.json.org/ lists more than two dozen Java libraries that have been created for processing JSON data. After conducting an informal review of web pages that discuss the various libraries, I decided to

concentrate on the json-simple library in this book. For example, here is the conclusion from the article titled A Review of 5 Java JSON Libraries :

Note:
"If you are looking for a simple lightweight Java library that reads and writes JSON, and supports Streams, JSON.simple is probably a good match. It does what it says on the box in 12 classes, and works on legacy (1.4) JREs. "

This conclusion is similar to conclusions that I found on several other websites. However, you should not take my selection of json-simple for this book as a recommendation for the json-simple library as compared to other libraries that are available. Once you have a basic understanding of how to process JSON using Java and the json-simple library, you should conduct your own review to identify the library that best suits your needs.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view Figure 1 while you are reading about it.

Background information

JSON is an acronym for **JavaScript Object Notation** . Don't be fooled by the name however. Although JSON is based on JavaScript object syntax, it is not JavaScript nor is it any programming language. As stated earlier, JSON is simply a *lightweight, text-based, language-independent data interchange format* -- nothing more and nothing less.

A lightweight text-based data interchange format

Similar to XML, JSON is a general purpose data interchange format that is supported by Java, PHP, JavaScript, and other programming languages. JSON is a standard that describes how ordered lists and unordered maps, strings, boolean values, and numbers can be represented as text in a string.

Similar to but less complex than XML, JSON provides a way to pass structured information between different computing environments using the same or different languages.

Typical operation

Typically a data construct, *(such as an object for example)* , in one programming environment will be transformed into a JSON string. That string will be transported to another programming environment where it will be transformed into a data construct, *(such as a hash table for example)* , that is suitable for use in that programming environment

A real-world analogy

Consider the following analogous situation. A young family has a large playscape for their children in their back yard. They need to move to another house across town. In order to save money, they rent a small truck and do the entire move themselves.

A playscape object

The playscape can be thought of as an object with certain properties such as **swing** and **slide** .

It is too large to fit into the truck so the adults disassemble it into a well-organized package of boards, chains, bolts, nuts, etc. They are very careful

to label each part and to create some drawings showing the organization of the parts for use later.

No longer an object

In that disassembled state, it can no longer be thought of as an object with properties of **swing** and **slide** . Instead, it is simply a well-organized and documented package of parts. The package of parts is analogous to a JSON string. The playscape object has been transformed into a well-organized package of parts.

Reassemble the parts

After the parts are transported to the new location, they are reassembled into an object with properties of **swing** and **slide** .

This is what we do with JSON. We disassemble an object *(or other data construct)* into a JSON string: a well-organized package of parts. Later on, and possibly in an entirely different programming environment, we reassemble the parts into a data construct suitable for use in the new programming environment.

Streamlined procedures

Java, JavaScript, PHP, and other programming languages provide streamlined procedures for transforming a data construct into a JSON string and for transforming a JSON string into a suitable data construct. As an example, the **toJASONString** method can be used to transform a Java object of type **JSONObject** into a JSON string. The **parse** method of the Java **JSONValue** class can be used to transform a JSON string into a **JSONObject** object. Other methods or functions are available to accomplish the same purposes in other languages such as JavaScript .

Online references

There are many good online JSON references. Here are a few:

- Introducing JSON
- The json-simple library
- JSON: The Fat-Free Alternative to XML
- Java API for JSON Processing: An Introduction to JSON
- JSON with PHP
- JSON tutorial for beginners learn how to program part 1 JavaScript (video)
- JSON in JavaScript
- JSON: What It Is, How It Works, How to Use It

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0200: The What and Why of JSON
- File: Json0200.htm
- Published: 05/29/16

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0200R: Review
This page contains review questions and answers for the page titled "Json0200 The What and Why of JSON" in the book titled "The json-simple Java Library".

Revised: Fri Jun 03 14:06:45 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of Contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page provides review questions and answers for the page titled Json0200 The What and Why of JSON . Once you study that page, you should be able to answer the review questions in this page.

The questions and the answers in this page are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

The Figures and Listings *(if any)* are grouped together. It is recommend that when a question or an answer refers to a Figure or a Listing, that you open it in a new window to make it easy to view it while reading the question or the answer.

Questions

Question 1 .

True or False? JSON is an acronym for "Java Object Names."

Go to answer 1

Question 2

True or False? JSON is based on a subset of the JavaScript Programming Language.

Go to answer 2

Question 3

True or False? JSON is a text format that is language independent but uses conventions that are familiar to programmers of the C-family of languages including C, C++, C#, Java, JavaScript, Perl, Python, Visual Basic, and many others.

Go to answer 3

Question 4

True or False? JSON is part of the JavaScript programming language.

Go to answer 4

Question 5

True or False? JSON can represent three structured types: objects, arrays, and conditionals.

Go to answer 5

Question 6

True or False? Figure 1 shows a JSON text string containing name/value pairs as well as nested arrays.

Go to answer 6

Question 7

True or False? JSON is important in the implementation of Big Data NoSQL databases such as MongoDB and Couchbase .

Go to answer 7

Question 8

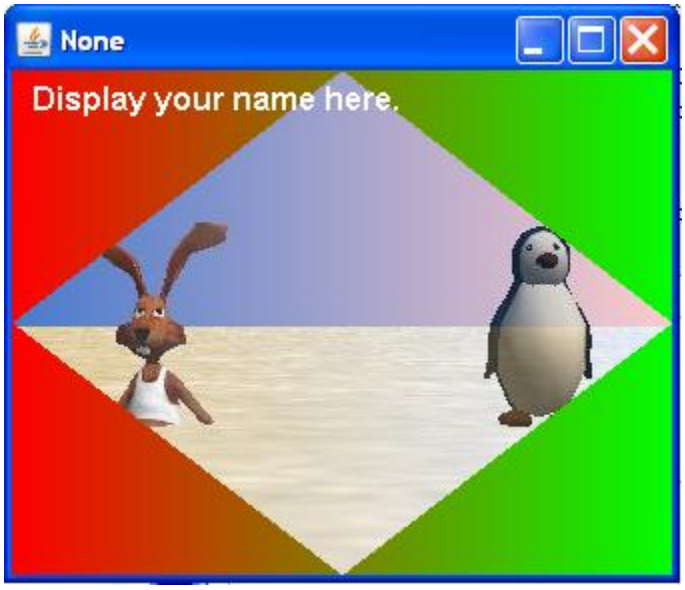
True or False? This book is intended to teach you about JSON.

Go to answer 8

Figure index

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 8

False. This book is not intended to teach you about JSON. There are numerous tutorials on the web that you can access for that purpose. This book is intended to teach you how to use the json-simple Java library, *(which is one of several available libraries)*, to parse, generate, transform, and query JSON.

Go back to Question 8

Answer 7

True. According to the InfoWorld article of August 25, 2014, "Several NoSQL databases -- including the wildly successful MongoDB and Couchbase -- store data in JSON documents natively."

Go back to Question 7

Answer 6

True.

Go back to Question 6

Answer 5

False. According to the online document titled Java API for JSON Processing: An Introduction to JSON , "JSON can represent two structured types: objects and arrays. An object is an unordered collection of zero or more name/value pairs. An array is an ordered sequence of zero or more values. The values can be strings, numbers, booleans, null, and these two structured types."

Go back to Question 5

Answer 4

False. It is important to note that even though JSON is based on JavaScript syntax, JASON is not JavaScript nor is it any other programming language. In fact, it is not a programming language at all. JSON is simply *"a lightweight, text-based, language-independent data exchange format"* -- nothing more and nothing less.

Go back to Question 4

Answer 3

False. Visual Basic is not based on the C-family of languages. Visual Basic is based on the BASIC programming language that came into existence in 1964. The C programming language came into existence independently in 1972.

Go back to Question 3

Answer 2

True.

Go back to Question 2

Answer 1

False. According to the online document titled Introducing JSON . "JSON (JavaScript Object Notation) is a ...".

Go back to Question 1

Figures

This section contains Figures that may be referred to by one or more questions or answers. These Figures may also be helpful as reference material for answering the questions.

Note: Figure 1.

```
{
  "game":
    [
      {"cards":
        [
```

```
          "2-club","3-heart","4-diamond","5-spade"
        ],
        "name":"Tom"
      },
      {"cards":
        [
          "4-heart","5-heart","6-club","7-diamond"
        ],
        "name":"Joe"
      }
    ]
}
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0200R: Review
- File: Json0200R.htm
- Published: 06/03/16

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0205: Getting Started
Learn how to download, install, and test the json-simple Java library.

Revised: Thu Jun 02 19:21:59 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page explains how to get started using the library.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

The json-simple library

Download the library

As of May 2016, the library can be downloaded in a jar file named json-simple-1.1.1.jar . The name of the jar file may be expected to change over time as new versions are released.

Library documentation

Also as of May 2016, a file named **json_simple-1.1-all.zip** , which contains source code for the library, can be downloaded from that same page. See the contents of that zip file for a copy of the license covering the material in the library.

I used the source code from that file to create a documentation package for version 1.1, which you can download here . Download and extract the contents of the file named **docs.zip** into an empty folder and open the file

named **index.html** in your browser to view the documentation in standard **javadoc** format.

Install the library

To use the library, you need to download the latest version of the jar file, store it somewhere on your system, and put it on your classpath. The instructions on this page apply to Windows 7. If you are using some other operating system, you will need to adjust the instructions to make them compatible with your operating system.

Test your installation

Create a Java source code file named **Code** containing the code shown in Listing 1 and store it in a folder somewhere on your system.

Note: Listing 1 . Java test program.

```
/**********************************************
***
Tested with Java 8, Windows 7, and
json-simple-1.1.1.jar
**********************************************
***/
import org.json.simple.ItemList;

class Code{
  public static void main(String[] args){
    System.out.println("ItemList: " + new
ItemList().

getClass().getSuperclass());
```

```
  }//end main
}//end class code
```

Assuming that you have downloaded the jar file and stored it in a folder named **json-simple** in the root directory of your C drive, create a batch file containing the commands shown in Listing 2 and store the batch file in the same folder as the source code file named **Code** .

Note: Listing 2 . Sample batch file.

```
echo off
rem Sets the classpath, compiles, and runs
Code.java.

del *.class

javac -cp .;C:\json-simple\json-simple-1.1.1.jar
Code.java

java -cp .;C:\json-simple\json-simple-1.1.1.jar
Code
pause
```

Double-click the batch file. Assuming that you have a compatible version of the Java Development Kit installed on your system, your test program should be compiled and executed, producing the output shown in Figure 1 on your computer screen.

Note: Figure 1. Output from test program.

```
ItemList: class java.lang.Object
Press any key to continue . . .
```

By way of explanation, the code in Listing 1 instantiates a new object of the class named **ItemList** from the JSON library. Then it gets and displays the superclass of that class, which is the class named **Object** . In other words, the JSON library class named **ItemList** extends the standard class named **Object** .

If you were able to produce the output shown in Figure 1 , you are ready to start generating, transforming, and querying JSON text using Java and the **json-simple** Java library.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0205: Getting Started
- File: Json0205.htm
- Published: 05/30/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you

should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.
In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0205R: Review
This page contains review questions and answers for the page titled "Json0205 Getting Started" in the book titled "The json-simple Java Library".

Revised: Fri Jun 03 14:54:03 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of Contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page provides review questions and answers for the page titled Json0205: Getting Started . Once you study that page, you should be able to answer the review questions in this page.

The questions and the answers in this page are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

Questions

Question 1 .

True or False? To use the library, you need to download the latest version of the jar file, store it somewhere on your system, and put it on your operating system's path environment variable.

Go to answer 1

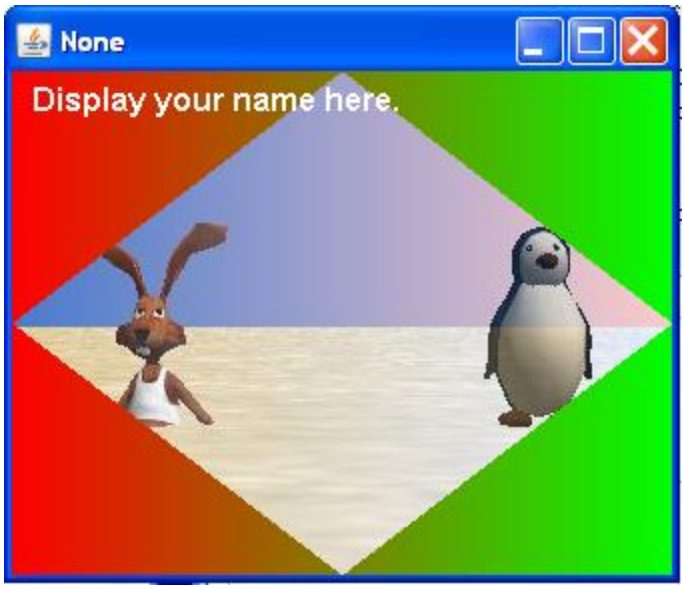
Question 2

True or False? The **json-simple** library class named **ItemList** extends the standard Java class named **ArrayList** .

Go to answer 2

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 2

False. The **json-simple** library class named **ItemList** extends the standard Java class named **Object** .

Go back to Question 2

Answer 1

False. To use the library, you need to download the latest version of the jar file, store it somewhere on your system, and put it on your operating system's ***classpath*** environment variable.

Go back to Question 1

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0205R: Review
- File: Json0205R.htm
- Published: 06/03/16

nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0210: Structure of the json-simple Java Library
This page explains the inheritance structure of the json-simple Java library and how it fits into the standard Java library.

Revised: Thu Jun 02 19:23:22 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page explains the inheritance structure of the **json-simple** Java library and how it fits into the standard Java library.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

General background information

The earlier page titled Json0205: Getting Started provided a download link for a standard **javadoc** documentation package for version 1.1 of the **json-simple** Java library. That documentation package shows that the library contains the following packages:

- org.json.simple
- org.json.simple.parser

Those packages contain the following classes and interfaces with the interfaces **shown in Italics** :

- *ContainerFactory*
- *ContentHandler*
- ItemList
- JSONArray
- *JSONAware*
- JSONObject
- JSONParser
- *JSONStreamAware*

- JSONValue
- ParseException
- Yytoken

Note:
The jar file also contains a class named **Yylex** in the **org.json.simple.parser** package that did not show up in the documentation. It didn't show up in the documentation because it is *package-private* and therefore not intended for our direct use. It is used internally by code in the class named **JSONParser** .

Discussion and sample code

Listing 1 provides the code for a Java program that establishes the position in the class hierarchy of each of the classes in the **json-simple** library by displaying the superclass of each of those classes. Figure 1 shows the screen output produced by this program.

Note: Figure 1. Inheritance hierarchy.

```
ItemList: class java.lang.Object
JSONArray: class java.util.ArrayList
JSONObject: class java.util.HashMap
JSONParser: class java.lang.Object
JSONValue: class java.lang.Object
ParseException: class java.lang.Exception
Yytoken: class java.lang.Object
```

The class named **ParseException** extends the standard class named **Exception** as you might expect.

Four of the classes extend the class named **Object** , which is not particularly surprising. New classes extend the **Object** class by default unless they are defined to purposely extend some other class.

The most important thing shown in Figure 1 is that the **JSONArray** class extends the standard **ArrayList** class and the **JSONObject** class extends the standard **HashMap** class. These standard classes are part of the *Java Collections Framework* .

This illustrates one of the most important features of the **json-simple** library. The library maximizes the use of standard classes from the *Java Collections Framework* . Once you have a reference to an object of the **JSONArray** class or an object of the **JSONObject** class, you have access to all of the polymorphic features provided by that framework. You also have access to the methods defined by those two classes and the classes defined by their superclasses. Therefore, if you are already skilled at programming within the collections framework, it is a small step to add JSON programming to your skill set.

What you can expect

This book will explain how to use the following classes:

- JSONObject
- JSONValue
- JSONParser
- JSONArray
- ParseException

An investigation into the use of the other classes and interfaces will be left as *"an exercise for the student"* .

Run the program

I encourage you to copy the code from Listing 1 . Execute the code and confirm that you get the same results as those shown in Figure 1 . Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Complete program listing

Listing 1 provides the source code for the program named **Code.java** .

Note: Listing 1 . Program named Code.java.

```
/*************************************************
******************
Copyright 2016, R.G.Baldwin
Establishes the position of each of the json-
simple classes in the
 class hierarchy by getting and displaying the
superclass of each
 class.

Produces the following output:

ItemList: class java.lang.Object
JSONArray: class java.util.ArrayList
JSONObject: class java.util.HashMap
JSONParser: class java.lang.Object
JSONValue: class java.lang.Object
ParseException: class java.lang.Exception
Yytoken: class java.lang.Object

Tested with Java 8, Windows 7, and json-simple-
1.1.1.jar
*************************************************
```

```
******************/
import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.JSONValue;
import org.json.simple.ItemList;
import org.json.simple.JSONAware;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import org.json.simple.parser.Yytoken;


class Code{
  public static void main(String[] args){

    System.out.println("ItemList: " +
                            new
ItemList().getClass().getSuperclass());
    System.out.println("JSONArray: " +
                           new
JSONArray().getClass().getSuperclass());
    System.out.println("JSONObject: " +
                           new
JSONObject().getClass().getSuperclass());
    System.out.println("JSONParser: " +
                           new
JSONParser().getClass().getSuperclass());
    System.out.println("JSONValue: " +
                            new
JSONValue().getClass().getSuperclass());
    System.out.println("ParseException: " +
             new
ParseException(ParseException.ERROR_UNEXPECTED_CHA
R).

getClass().getSuperclass());
    System.out.println("Yytoken: " +
                                    new
```

```
Yytoken(Yytoken.TYPE_COLON,":").

getClass().getSuperclass());
  }//end main
}//end class code
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0210: Structure of the json-simple Java Library
- File: Json0210.htm
- Published: 05/30/16

on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0210R: Review
This page contains review questions and answers for the page titled "Json0210: Structure of the json-simple Java Library" in the book titled "The json-simple Java Library".

Revised: Fri Jun 03 17:21:17 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of Contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page provides review questions and answers for the page titled Json0210: Structure of the json-simple Java Library . Once you study that page, you should be able to answer the review questions in this page.

The questions and the answers in this page are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

Questions

Question 1 .

True or False? Version 1.1 of the **json-simple** Java library contains the following packages:

- org.json.simple
- org.json.simple.parser

Go to answer 1

Question 2

True or False? Version 1.1 of the **json-simple** Java library contains the public classes shown below

- ItemList
- JSONArray
- JSONObject
- JSONParser
- JSONValue
- ParseException
- Object
- Yytoken

Go to answer 2

Question 3

True or False? Version 1.1 of the **json-simple** Java library contains a class named **Yylex** in the **org.json.simple.parser** package that is declared to be *package-private* .

Go to answer 3

Question 4

True or False? Despite its name, the **JSONArray** class is not really an array. Instead, it is a **List** .

Go to answer 4

Question 5

True or False? Objects of the **JSONArray** class contain *unordered* data.

Go to answer 5

Question 6

True or False? Other than the **JSONObject** class and the **JSONArray** class, all of the classes in version 1.1 of the **json-simple** Java library extend the **Object** class.

Go to answer 6

Question 7

True or False? The **JSONObject** class extends the standard Java **TreeSet** class.

Go to answer 7

Question 8

True or False? An object of the **JSONObject** class maps *keys* or *names* to *values* .

Go to answer 8

Question 9

True or False? An object of the **JSONObject** class can contain duplicate keys but cannot contain duplicate values.

Go to answer 9

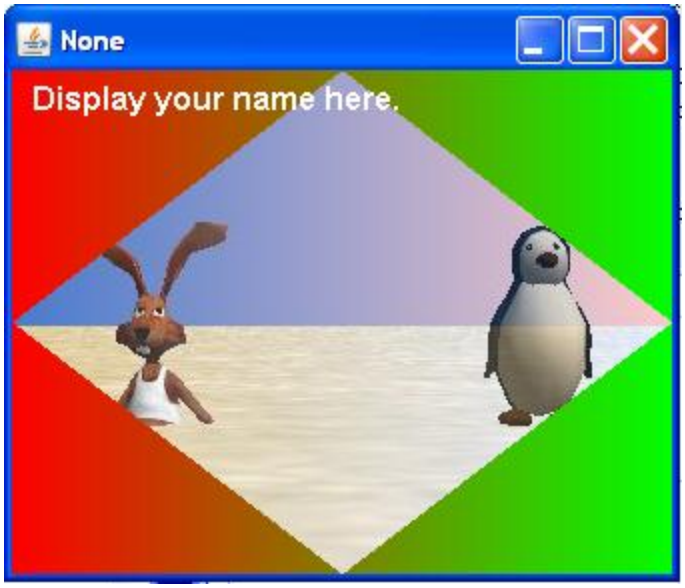
Question 10

True or False? An object of the **JSONObject** class is an unordered collection.

Go to answer 10

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 10

True. An object of the **JSONObject** class extends the **HashMap** class and implements the **Map** interface. According to the Oracle documentation, *"The order of a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the **TreeMap** class, make specific guarantees as to their order; others, like the **HashMap** class, do not."*

Go back to Question 10

Answer 9

False. The **JSONObject** class implements the **Map** interface. According to the Oracle documentation, *"A map cannot contain duplicate keys; each key can map to at most one value."*

Go back to Question 9

Answer 8

True. JSON documentation often speaks of *name/value* pairs. However, the **JSONObject** class extends the standard Java **HashMap** class, which implements the **Map** interface. According to the Oracle Java documentation, an object that implements the **Map** interface is *"An object that maps keys to values."* Therefore, it is probably acceptable to use the terms *name/value* and *key/value* interchangeably.

Go back to Question 8

Answer 7

False. The **JSONObject** class extends the standard Java **HashMap** class.

Go back to Question 7

Answer 6

True.

Go back to Question 6

Answer 5

False. Because the **JSONArray** class extends the **ArrayList** class, which implements the **List** interface, it is *"An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list."*

Go back to Question 5

Answer 4

True. The **JSONArray** class extends the standard **ArrayList** class, which implements the **List** interface.

Go back to Question 4

Answer 3

True.

Go back to Question 3

Answer 2

False. Version 1.1 of the **json-simple** Java library does not contain a class named **Object** .

Go back to Question 2

Answer 1

True.

Go back to Question 1

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0210R: Review
- File: Json0210R.htm
- Published: 06/03/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0215: Encoding JSON Strings
Learn how to use the JSONObject class to encode key/value pairs into JSON strings.

Revised: Thu Jun 02 19:24:40 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page explains how to use the **JSONObject** class to encode key/value pairs into JSON strings.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

General background information

As you learned in the earlier page titled Json0210: Structure of the json-simple Java Library , the class named **JSONObject** extends the standard Java class named **HashMap** . Therefore, once you have an object of the class named **JSONObject** , you can call any of the methods defined in or inherited into the **JSONObject** class on that object.

In the sample program that follows, we will use the **put** method that is inherited from the **HashMap** class to populate the object with key/value pairs. We will use the **toJSONString** method that is defined in the **JSONObject** class to transform the populated object into a JSON string.

Discussion and sample code

The program named **Code** *(see Listing 6)* constructs three **JSONObject** objects, populates the objects with key/value pairs, and saves the **JSONObject** objects in an **ArrayList** object. *(Actually it saves references to the **JSONObject** objects in the **ArrayList** object.)*

Then it transforms each **JSONObject** object into a JSON string. At this point, the JSON strings could be written to an output stream and transferred to a different programming environment. However, to keep the program simple, the program simply saves the JSON strings in a second **ArrayList** object for later display.

Then the program displays the JSON strings for comparison with the code that populated the **JSONObject** objects in the first place.

Create ArrayList containers

A complete listing of the program named **Code** is provided in Listing 6 near the end of the page. I will discuss and explain the code in fragments. The first fragment is shown in Listing 1 .

Note: Listing 1 . Create ArrayList containers.

```
import org.json.simple.JSONObject;
import java.util.ArrayList;
import java.util.Iterator;
```

```
class Code {

  public static void main(String[] args){
    //Create a container for several JSON objects.
    ArrayList <JSONObject> listA = new
ArrayList<JSONObject>();
    //Create a container for several JSON strings
    ArrayList <String> listB = new
ArrayList<String>();
```

The code in Listing 1 instantiates two objects of the class named **ArrayList** to serve as containers for the **JSONObject** objects and the JSON strings. This is plain vanilla Java code. There is nothing new here.

Create and populate a JSONObject object

Listing 2 creates and populates the first **JSONObject** object with key/value pairs. The keys are "name", "age", and "student" respectively. The value types are **String** , **int** , and **boolean** respectively.

Note: Listing 2 . Create and populate a JSONObject object.

```
    listA.add(new JSONObject());

    listA.get(0).put("name","Joe");
    listA.get(0).put("age",21);
    listA.get(0).put("student",true);
```

The code in Listing 2 begins by instantiating a new object of the **JSONObject** class and adding its reference into the first element *(0)* of the **ArrayList** object. Then it calls the **get** method on the **ArrayList** object three times in succession to gain access to the **JSONObject** . Each time it gains access to the **JSONObject** object, it calls the **put** method inherited from the **HashMap** class to store a key/value pair in the **JSONObject** object.

Note:
Note that the data stored in the **JSONObject** object is an unordered collection. As you will see later, the order in which the key/value pairs are extracted from the object using an iterator is unrelated to the order in which the key/value pairs are stored in the object.

Create and populate two more JSONObject objects

Listing 3 repeats the process two more times to create, populate, and save two more **JSONObject** objects. Note that one of these objects is populated in a different order than is the case in Listing 2 .

Note: Listing 3 . Create and populate two more JSONObject objects.

```
//Create and populate the second JSONObject.
listA.add(new JSONObject());
listA.get(1).put("student",false);
listA.get(1).put("name","Sue");
listA.get(1).put("age",32);


//Create and populate the third JSONObject
listA.add(new JSONObject());
```

```
    listA.get(2).put("name","Tom");
    listA.get(2).put("age",19);
    listA.get(2).put("student",true);
```

Encode the data into JSON strings

Listing 4 uses an **Iterator** to gain access to each populated **JSONObject** object. Each time it gains access to an object, it calls the **toJSONString** method that is defined in the **JSONObject** class to transform the object into a JSON string.

At this point, the program could write the JSON strings into an output stream for transfer to some other programming environment. However as mentioned earlier, to keep the program simple, the program saves the JSON strings as elements in a second **ArrayList** object.

Note: Listing 4 . Encode the data into JSON strings.

```
    Iterator<JSONObject> iteratorA =
listA.iterator();
    while (iteratorA.hasNext()){
      listB.add(iteratorA.next().toJSONString());
    }//end while loop
```

Display the JSON strings

Listing 5 uses an **Iterator** to access and display each of the JSON strings that are stored in the **ArrayList** object.

Note: Listing 5 . Display the JSON strings.

```
    Iterator<String> iteratorB = listB.iterator();
    while (iteratorB.hasNext()){
      System.out.println(iteratorB.next());
    }//end while loop

  }//end main
}//end class Code
```

The screen output is shown in Figure 1 .

Note: Figure 1. The screen output.

```
{"student":true,"name":"Joe","age":21}
{"student":false,"name":"Sue","age":32}
{"student":true,"name":"Tom","age":19}
```

The format for each line of text that you see in Figure 1 is the format that you would expect for a JSON string that doesn't include array data. *(See Introducing JSON for more details regarding format.)*

Briefly, each key is separated from its value by a colon *(:)* . Each key/value pair is separated from its neighbors by a comma *(,)* . The JSON string begins with a left brace *({)* and ends with a right brace *(})* . Keys are surrounded by double quotation characters. Values may or may not be separated by double quotation characters, depending on their type. Some

characters inside of values, such as double quotation characters and backslash characters must be escaped with a backslash character *(\" and \\)* but that is not shown by this program.

Note that the display order for the key/value pairs shown in Figure 1 does not match the order in which the objects were populated in Listing 2 and Listing 3 . A HashMap does not impose an ordering on its contents and does not guarantee that the ordering will remain constant over time.

The end of the program

Listing 5 also signals the end of the **main** method and the end of the program.

Run the program

I encourage you to copy the code from Listing 6 . Execute the code and confirm that you get the same results as those shown in Figure 1 . Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Complete program listing

A complete listing of the program named Code.java is provided in Listing 6 .

Note: Listing 6. The program named Code.java.

```
/*********************************************************
******************
Copyright 2016 R.G.Baldwin
```

```
Constructs three JSONObject objects and saves them
in an ArrayList.
Transforms each JSONObject object into a String
object and saves the
 strings in a second ArrayList object.
Displays the strings.

Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
**************************************************
******************/

import org.json.simple.JSONObject;
import java.util.ArrayList;
import java.util.Iterator;

class Code {

  public static void main(String[] args){
    //Create a container for several JSON objects.
    ArrayList <JSONObject> listA = new
ArrayList<JSONObject>();
    //Create a container for several JSON strings
    ArrayList <String> listB = new
ArrayList<String>();

    //Create and populate the first JSONObject
with unordered
    // key/value pairs.
    listA.add(new JSONObject());
    listA.get(0).put("name","Joe");
    listA.get(0).put("age",21);
    listA.get(0).put("student",true);

    //Create and populate the second JSONObject.
Note that the object
```

```
    // is populated in a different order than
above.
    listA.add(new JSONObject());
    listA.get(1).put("student",false);
    listA.get(1).put("name","Sue");
    listA.get(1).put("age",32);


    //Create and populate the third JSONObject
    listA.add(new JSONObject());
    listA.get(2).put("name","Tom");
    listA.get(2).put("age",19);
    listA.get(2).put("student",true);

    //Transform the three JSON objects into JSON
strings and save
    // them in ListB. Could write them to disk for
transfer to a
    // different programming environment at this
point.
    Iterator<JSONObject> iteratorA =
listA.iterator();
    while (iteratorA.hasNext()){
      listB.add(iteratorA.next().toJSONString());
    }//end while loop

    //Display the JSON strings currently stored in
listB. Note that
    // the display order does not necessarily
match the order in
    // which the original objects were populated.
    Iterator<String> iteratorB = listB.iterator();
    while (iteratorB.hasNext()){
      System.out.println(iteratorB.next());
    }//end while loop
```

```
  }//end main
}//end class Code
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0215: Encoding JSON Strings
- File: Json0215.htm
- Published: 05/31/16

-end-

Json0215R: Review
This page contains review questions and answers for the page titled "Json0215: Encoding JSON Strings" in the book titled "The json-simple Java Library".

Revised: Sat Jun 04 12:48:39 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of Contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page provides review questions and answers for the page titled Json0215: Encoding JSON Strings . Once you study that page, you should be able to answer the review questions in this page.

The questions and the answers in this page are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

The figures and listings *(if any)* are grouped together. It is recommend that when a question or an answer refers to a figure or a listing, that you open it in a new window to make it easy to view it while reading the question or the answer.

Questions

Question 1 .

True or False? Once you have an object of the class named **JSONObject** , you can call any of the methods defined in the **HashMap** class on that object.

Go to answer 1

Question 2

True or False? The code shown in Listing 1 produced the screen output shown in Figure 1 .

Go to answer 2

Question 3

True or False? The code shown in Listing 3 produced the screen output shown in Figure 3 .

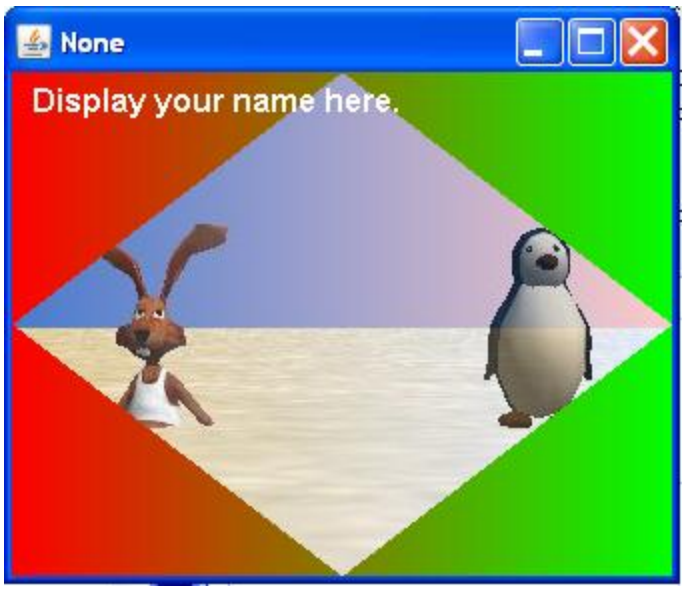
Go to answer 3

Figure index

Listing index

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 3

True. The code in Listing 3 uses the **get** method inherited from the **HashMap** class to get and display the values associated with the keys "name", "age", and "student".

Go back to Question 3

Answer 2

False. The code in Listing 1 produced the output with the compiler error message shown in Figure 2 . The output shown in in Figure 1 was produced by the code shown in Listing 2 . Note the call to the **toJSONString** method of the **JSONObject** class to extract the information from the **JSONObject** object and encode it into a JSON string.

Go back to Question 2

Answer 1

True. The class named **JSONObject** extends the standard Java class named **HashMap** . Therefore, the **JSONObject** class inherits all of the methods defined in the **HashMap** class.

Go back to Question 1

Figures

This section contains Figures that may be referred to by one or more questions or answers.

Note:
Figure 1

```
{"student":true,"name":"Joe","age":21}
```

Note:
Figure 2

```
Code01.java:20: error: no suitable method found
for get(no arguments)
    String jsonString = jsonObject.get();
```

Note:
Figure 3

```
Joe
21
true
```

Listings

This section contains Listings that may be referred to by one or more questions or answers.

Note:
Listing 1

```
import org.json.simple.JSONObject;

class Code01 {

  public static void main(String[] args){

    JSONObject jsonObject= new JSONObject();
    jsonObject.put("name","Joe");
    jsonObject.put("age",21);
    jsonObject.put("student",true);

    String jsonString = jsonObject.get();
    System.out.println(jsonString);

  }//end main
}//end class Code01
```

Note:
Listing 2

```
import org.json.simple.JSONObject;

class Code02 {

  public static void main(String[] args){

    JSONObject jsonObject= new JSONObject();
    jsonObject.put("name","Joe");
    jsonObject.put("age",21);
    jsonObject.put("student",true);

    String jsonString = jsonObject.toJSONString();
    System.out.println(jsonString);

  }//end main
}//end class Code02
```

Note:
Listing 3

```
import org.json.simple.JSONObject;

class Code03 {

  public static void main(String[] args){
```

```
    JSONObject jsonObject= new JSONObject();
    jsonObject.put("name","Joe");
    jsonObject.put("age",21);
    jsonObject.put("student",true);

    System.out.println(jsonObject.get("name"));
    System.out.println(jsonObject.get("age"));
    System.out.println(jsonObject.get("student"));

  }//end main
}//end class Code03
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0215R: Review
- File: Json0215R.htm
- Published: 06/04/16

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0220: Decoding JSON Strings
Learn to decode JSON strings using the parse methods of the JSONValue and JSONParser classes.

Revised: Thu Jun 02 19:25:56 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page explains how to decode JSON strings using the parse methods of the **JSONValue** and **JSONParser** classes.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

General background information

As mentioned in the Preface , this page deals with the classes named **JSONValue** and **JSONParser** . Each of these classes extends the **Object** class and provides various methods for processing JSON data. The page also exposes you to the class named **ParseException** .

All of the methods in the **JSONValue** class are **static** methods. Methods in this class are available to deal with JSON input data in both the string format and the **JSONObject** format.

None of the methods in the **JSONParser** class are static methods. In general the methods in this class are designed to deal only with input data in the string format.

Both classes provide several overloaded **parse** methods that can be used to parse input JSON text from different sources into the **JSONObject** format.

The sample program that follows will use the **parse** method from both classes to parse JSON strings into **JSONObject** objects.

Discussion and sample code

The program named **Code** *(see Listing 6)* consists of some old code and some new code. The program begins just like the program that I explained in the earlier page titled Json0215: Encoding JSON Strings . The beginning portion of the program is used solely to create JSON data in string format and to store the individual strings as elements in an **ArrayList** object.

This code constructs three **JSONObject** objects and saves them in an **ArrayList** object. Then it transforms each **JSONObject** object into a **String** object and saves the strings in a second **ArrayList** object. The new code decodes the strings into **JSONObject** objects using a **parse** method from the **JSONValue** class and saves them in a third **ArrayList** object.

Then the new code decodes the strings into **JSONObject** objects using a **parse** method from the **JSONParser** class and saves them in a fourth **ArrayList** object.

The contents of both lists of **JSONObject** objects are displayed on the computer screen after they are populated with decoded data from the JSON strings.

Decode and display using the JSONValue class

I will explain this program in fragments, and will begin at the point where the new code begins. The first fragment is shown in Listing 1 .

Note: Listing 1 . Decode and display using the JSONValue class.

```
    //Create a container for decoded JSON strings.
    ArrayList <JSONObject> listC = new
ArrayList<JSONObject>();

    //Decode and display JSON strings using the
parse method of the
    // JSONValue class
    System.out.println("Decode using JSONValue
class");
    decodeC(listB,listC);
    display(listC);
    System.out.println();//blank line
```

Immediately prior to the code in Listing 1 , three JSON strings are stored as elements in an **ArrayList** object referred to as **listB** .

Listing 1 begins by instantiating a new **ArrayList** object, referred to as **listC** that will receive the decoded versions of the JSON strings as type **JSONObject** objects. Then Listing 1 calls the method named **decodeC** passing the list of JSON strings and the empty list as parameters.

The method named decodeC

The method named **decodeC** is shown in its entirety in Listing 2 .

Note: Listing 2 . The method named decodeC.

```
  static void decodeC(ArrayList input,ArrayList
output){
    String temp = null;
    Iterator<String> iterator = input.iterator();
    while (iterator.hasNext()){
      temp = iterator.next();
      output.add(JSONValue.parse(temp));
    }//end while loop
  }//end decodeC
```

This method decodes a list of JSON strings into **JSONObject** objects using a static **parse** method of the **JSONValue** class. The resulting **JSONObject** objects are added to the empty **ArrayList** object received as an incoming parameter. When the method returns, that list contains one **JSONObject** object for each JSON string contained in the incoming **ArrayList** object.

Display contents of JSONObject objects

Returning now to the code in Listing 1 , the next statement calls the **display** method passing the now-populated **listC** as a parameter. At this point, the **JSONObject** objects stored in **listC** contain the information that was extracted from the JSON strings by the **parse** method of the **JSONValue** class.

The **display** method is shown in its entirety in Listing 3 .

Note: Listing 3 . The display method.

```
  static void display(ArrayList input){
    JSONObject temp = null;
    Iterator<JSONObject> iterator =
input.iterator();
    while (iterator.hasNext()){
      temp = iterator.next();
      System.out.print("Name: " +
temp.get("name"));
      System.out.print("  Age: " +
temp.get("age"));
      System.out.println("  Is student? " +
temp.get("student"));
    }//end while loop
  }//end display
```

This method uses the **get** method inherited from the **HashMap** class to get and display the values in a list of **JSONObject** objects for a known set of keys. This is plain-vanilla Java code and shouldn't require further explanation. The results of this call to the **display** method are shown in the top half of Figure 1 .

Note: Figure 1. Program output.

```
Decode using JSONValue class
Name: Joe Age: 21 Is student? true
Name: Sue Age: 32 Is student? false
Name: Tom Age: 19 Is student? true

Decode using JSONParser class
```

```
Name: Joe Age: 21 Is student? true
Name: Sue Age: 32 Is student? false
Name: Tom Age: 19 Is student? true
```

The contents of the JSON strings for this same data were displayed in Figure 1 of the earlier page titled Json0215: Encoding JSON Strings .

If you compare the output in the top half of Figure 1 above with the JSON string data on the earlier page, you will see that they match.

Note:
You could also make the comparison with the data in the code in the early portion of Listing 6 .

Decode and display using the JSONParser class

Returning to the **main** method, the code in Listing 4 calls the **decodeD** method passing the list of JSON strings along with an empty **ArrayList** object for the purpose of decoding the JSON strings using a **parse** method of the **JSONParser** class.

Note: Listing 4 . Decode and display using the JSONParser class.

```
    //Create another container for decoded JSON
strings.
    ArrayList <JSONObject> listD = new
ArrayList<JSONObject>();

    //Decode and display JSON strings using the
```

```
parse method of the
    // JSONParser class
    System.out.println("Decode using JSONParser
class");
    decodeD(listB,listD);
    display(listD);
    System.out.println();//blank line

  }//end main
```

Then the code in Listing 4 calls the **display** method to display the results, producing the text in the bottom half of Figure 1 .

As you can see, the results in the top and bottom halves of Figure 1 match, indicating that both approaches produced the same results.

The method named decodeD

The method named decodeD is shown in its entirety in Listing 5 .

Note: Listing 5 . The method named decodeD.

```
  static void decodeD(ArrayList input,ArrayList
output){
    JSONParser parser = new JSONParser();
    String temp = null;
    Iterator<String> iterator = input.iterator();
    while (iterator.hasNext()){
      temp = iterator.next();
      try{
        //ParseException; must be caught or
```

```
declared to be thrown.
        output.add(parser.parse(temp));
      }catch(ParseException pex){
        pex.printStackTrace();
      }//end catch
    }//end while loop
  }//end decodeD
```

This method decodes a list of JSON strings into **JSONObject** objects using a **parse** method of the **JSONParser** class. This code is only slightly more complicated than the code shown earlier in Listing 2 that uses a **parse** method of the **JSONValue** class.

The additional complexity is due mainly to the fact that the **parse** method throws a *checked* exception named **ParseException** . Because it is a checked exception, it must either be caught or declared to be thrown by the method. I elected to catch it in this program and to simply call the **printStackTrace** method *(inherited from the **Throwable** class)* in the catch block. However, the **ParseException** class provides some other methods that can be called to elaborate on the nature of the error if desired.

The end of the program

Returning to the **main** method in Listing 4 , there is nothing more to do, so the last line in Listing 4 signals the end of the program.

Run the program

I encourage you to copy the code from Listing 6 . Execute the code and confirm that you get the same results as those shown in Figure 1 . Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Complete program listing

A complete listing of the program named **Code** is provided in Listing 6 .

Note: Listing 6 . The program named Code.java.

```
/*********************************************************
******************
Copyright 2016 R.G.Baldwin

OLD CODE:
Constructs three JSONObject objects and saves them
in an ArrayList.
Transforms each JSONObject object into a String
object and saves the
 strings in a second ArrayList object.

NEW CODE:
Decodes the strings into JSONObjects using the
JSONValue parse method
 and saves them in a third ArrayList object.
Decodes the strings into JSONObjects using the
JSONParser class
 and saves them in a fourth ArrayList object.
Displays both sets of decoded JSON strings.

Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
*********************************************************
*****************/

import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
import org.json.simple.parser.JSONParser;
```

```
import org.json.simple.parser.ParseException;
import java.util.ArrayList;
import java.util.Iterator;

class Code {

  public static void main(String[] args){
    //OLD CODE:
    //Create a container for several JSON objects.
    ArrayList <JSONObject> listA = new
ArrayList<JSONObject>();
    //Create a container for several JSON strings
    ArrayList <String> listB = new
ArrayList<String>();

    //Create and populate the first JSONObject
with unordered
    // key/value pairs.
    listA.add(new JSONObject());
    listA.get(0).put("name","Joe");
    listA.get(0).put("age",21);
    listA.get(0).put("student",true);

    //Create and populate the second JSONObject.
Note that the object
    // is populated in a different order than
above.
    listA.add(new JSONObject());
    listA.get(1).put("student",false);
    listA.get(1).put("name","Sue");
    listA.get(1).put("age",32);


    //Create and populate the third JSONObject
    listA.add(new JSONObject());
    listA.get(2).put("name","Tom");
    listA.get(2).put("age",19);
```

```
    listA.get(2).put("student",true);

    //Transform the three JSON objects into JSON
strings and save
    // them in ListB. Could write them to disk for
transfer to a
    // different programming environment at this
point.
    Iterator<JSONObject> iteratorA =
listA.iterator();
    while (iteratorA.hasNext()){
      listB.add(iteratorA.next().toJSONString());
    }//end while loop

    //NEW CODE BEGINS HERE

    //Create a container for decoded JSON strings.
    ArrayList <JSONObject> listC = new
ArrayList<JSONObject>();

    //Decode and display JSON strings using the
parse method of the
    // JSONValue class
    System.out.println("Decode using JSONValue
class");
    decodeC(listB,listC);
    display(listC);
    System.out.println();//blank line

    //Create another container for decoded JSON
strings.
    ArrayList <JSONObject> listD = new
ArrayList<JSONObject>();

    //Decode and display JSON strings using the
parse method of the
    // JSONParser class
```

```
    System.out.println("Decode using JSONParser
class");
    decodeD(listB,listD);
    display(listD);
    System.out.println();//blank line

  }//end main
  //-------------------------------------------
-----------------//

  //Decode a list of JSON strings into JSONObject
objects using the
  // parse method of the JSONValue class
  static void decodeC(ArrayList input,ArrayList
output){
    String temp = null;
    Iterator<String> iterator = input.iterator();
    while (iterator.hasNext()){
      temp = iterator.next();
      output.add(JSONValue.parse(temp));
    }//end while loop
  }//end decodeC
  //-------------------------------------------
-----------------//

  //Decode a list of JSON strings into JSONObject
objects using the
  // parse method of the JSONParser class
  static void decodeD(ArrayList input,ArrayList
output){
    JSONParser parser = new JSONParser();
    String temp = null;
    Iterator<String> iterator = input.iterator();
    while (iterator.hasNext()){
      temp = iterator.next();
      try{
        //ParseException; must be caught or
```

```
declared to be thrown.
        output.add(parser.parse(temp));
      }catch(ParseException pex){
        pex.printStackTrace();
      }//end catch
    }//end while loop
  }//end decodeD
  //-------------------------------------------
-----------------//

  //Display the values in a list of JSONObject
objects for a known
  // set of keys.
  static void display(ArrayList input){
    JSONObject temp = null;
    Iterator<JSONObject> iterator =
input.iterator();
    while (iterator.hasNext()){
      temp = iterator.next();
      System.out.print("Name: " +
temp.get("name"));
      System.out.print("  Age: " +
temp.get("age"));
      System.out.println("  Is student? " +
temp.get("student"));
    }//end while loop
  }//end display
  //-------------------------------------------
-----------------//
}//end class Code
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0220: Decoding JSON Strings
- File: Json0220.htm
- Published: 05/31/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0220R: Review
This page contains review questions and answers for the page titled "Json0220: Decoding JSON Strings" in the book titled "The json-simple Java Library".

Revised: Sat Jun 04 16:17:01 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of Contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page provides review questions and answers for the page titled Json0220: Decoding JSON Strings . Once you study that page, you should be able to answer the review questions in this page.

The questions and the answers in this page are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

When a question or an answer provides a link to a figure or a listing, you should open that link in a new window to make it easy to view while reading the question or the answer.

Note:
NOTE:
With respect to the screen output shown on this page, ignore the possibility of output or lack of output similar to the following:

```
Note: Code99.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
```

Questions

Question 1 .

True or False? The only way to decode or parse a JSON string with the **json-simple** library is by using the **parse** method belonging to an object of the class named **JSONParser** .

Go to answer 1

Question 2

True or False? Decoding or parsing a JSON string with the **json-simple** library requires the use of the **parse** method belonging to either an object of the class named **JSONParser** or an object of the class named **JSONValue** .

Go to answer 2

Question 3

True or False? The **parse** method belonging to the class named **JSONParser** can be called without the requirement to instantiate an object of the class.

Go to answer 3

Question 4

True or False? The **parse** methods of the **JSONValue** and **JSONParser** classes return an object of class **JSONString** .

Go to answer 4

Question 5

True or False? The code in Listing 1 produces the screen output shown in Figure 1 .

Go to answer 5

Question 6

True or False? The code in Listing 2 produces the screen output shown in Figure 3 .

Go to answer 6

Question 7

True or False? The code in Listing 3 produces the screen output shown in Figure 5 .

Go to answer 7

Question 8

True or False? The code in Listing 4 produces the screen output shown in Figure 6 .

Go to answer 8

Question 9

True or False? The code in Listing 5 produces the screen output shown in Figure 7 .

Go to answer 9

Question 10

True or False? The code in Listing 6 produces the screen output shown in Figure 9 .

Go to answer 10

Question 11

True or False? The code in Listing 7 produces the screen output shown in Figure 11 .

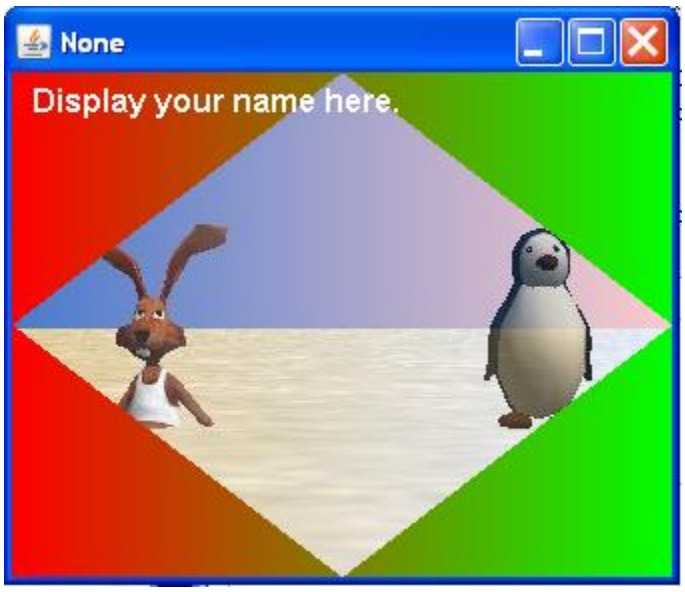
Go to answer 11

Figure index

Listing index

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 11

True.

Go back to Question 11

Answer 10

False. The code in Listing 6 produces the screen output with the errors shown in Figure 10 .

Go back to Question 10

Answer 9

False. The code in Listing 5 produces the screen output shown in Figure 8 . The **parse** method is not ***static*** in the **JSONParser** class.

Go back to Question 9

Answer 8

True.

Go back to Question 8

Answer 7

True. Even though it is not necessary to instantiate an object of a class to call a ***static*** method belonging to that class, it is possible to instantiate an object of the **JSONValue** class and to call the ***static*** **parse** method on that object.

Go back to Question 7

Answer 6

False. The code in Listing 2 produces the screen output with the compiler errors shown in Figure 4 . The return value from the **parse** method must be cast from type **Object** to type **JSONObject** to store it in a variable of type **JSONObject** .

Go back to Question 6

Answer 5

False. The code in Listing 1 produces the screen output with the compiler errors shown in Figure 2 . The **json-simple** library does not define a class named **JSONString** .

Go back to Question 5

Answer 4

False. The **json-simple** library does not define a class named **JSONString** . Both **parse** methods return a reference of type **java.lang.Object** . However, it is actually a reference to an object of the class **JSONObject** , which is a subclass of **Object** several levels down. The reference must be downcast to type **JSONObject** in order to call some of the methods defined in the **JSONObject** class or to store the object's reference in a variable of type **JSONObject** .

Go back to Question 4

Answer 3

False. The **parse** method belonging to the class named **JSONParser** is not declared ***static*** . Therefore, it is an *instance method* . Instance methods can only be called on an object instantiated from the class.

Go back to Question 3

Answer 2

False. The **parse** method that is defined in the class named **JSONValue** is a ***static*** method. Static methods belonging to a class can be called without the requirement to instantiate an object of the class.

Go back to Question 2

Answer 1

False. The class named **JSONValue** also defines a method named **parse** that can be used to decode or parse a JSON string.

Go back to Question 1

Figures

This section contains Figures that may be referred to by one or more questions or answers.

Note:
Figure 1

```
Joe
21
```

Note:
Figure 2

```
Code01.java:8: error: cannot find symbol
import org.json.simple.JSONString;
                      ^
  symbol:   class JSONString
  location: package org.json.simple
Code01.java:20: error: cannot find symbol
    JSONString jsonString02 =
JSONValue.parse(jsonString0101);
    ^
  symbol:   class JSONString
  location: class Code01
Code01.java:20: error: cannot find symbol
```

```
    JSONString jsonString02 =
JSONValue.parse(jsonString0101);
                                                  ^
  symbol:   variable jsonString0101
  location: class Code01
Note: Code01.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
3 errors
Error: Could not find or load main class Code01
```

Note:
Figure 3

```
Joe
21
```

Note:
Figure 4

```
Code02.java:19: error: incompatible types: Object
cannot be converted to JSONObj
ect
    JSONObject jsonObj02 =
JSONValue.parse(jsonString);
                                                ^
Note: Code02.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
```

```
1 error
Error: Could not find or load main class Code02
```

Note:
Figure 5

```
Joe
21
```

Note:
Figure 6

```
Joe
21
```

Note:
Figure 7

```
Joe
21
```

Note:
Figure 8

```
Code05.java:19: error: non-static method
parse(String) cannot be referenced from
 a static context
    JSONObject jsonObj02 =
(JSONObject)JSONParser.parse(jsonString);
                                        ^
Note: Code05.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
Error: Could not find or load main class Code05
```

Note:
Figure 9

```
Joe
21
```

Note:
Figure 10

```
Code06.java:19: error: unreported exception
ParseException; must be caught or de
clared to be thrown
    JSONObject jsonObj02 = (JSONObject)(new
JSONParser().parse(jsonString));

^
Note: Code06.java uses unchecked or unsafe
```

```
operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
Error: Could not find or load main class Code06
```

Note:
Figure 11

```
Joe
21
```

Listings

This section contains Listings that may be referred to by one or more questions or answers.

Note:
Listing 1

```
/*********************************************************
******************
Copyright 2016 R.G.Baldwin
Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
*********************************************************
******************/

import org.json.simple.JSONObject;
```

```
import org.json.simple.JSONValue;
import org.json.simple.JSONString;

class Code01 {

  public static void main(String[] args){

    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("name","Joe");
    jsonObj01.put("age",21);

    String jsonString01 =
jsonObj01.toJSONString();

    JSONString jsonString02 =
JSONValue.parse(jsonString0101);
    System.out.println(jsonString02.get("name"));
    System.out.println(jsonString02.get("age"));

  }//end main

}//end class Code01
```

Note:
Listing 2

```
/*************************************************
******************
Copyright 2016 R.G.Baldwin
Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
*************************************************
******************/
```

```
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;

class Code02 {

  public static void main(String[] args){

    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("name","Joe");
    jsonObj01.put("age",21);

    String jsonString = jsonObj01.toJSONString();

    JSONObject jsonObj02 =
JSONValue.parse(jsonString);
    System.out.println(jsonObj02.get("name"));
    System.out.println(jsonObj02.get("age"));

  }//end main

}//end class Code02
```

Note:
Listing 3

```
/*************************************************
******************
Copyright 2016 R.G.Baldwin
Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
*************************************************
******************/
```

```
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;

class Code03 {

  public static void main(String[] args){

    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("name","Joe");
    jsonObj01.put("age",21);

    String jsonString = jsonObj01.toJSONString();

    JSONObject jsonObj02 =
                      (JSONObject)(new
JSONValue().parse(jsonString));
    System.out.println(jsonObj02.get("name"));
    System.out.println(jsonObj02.get("age"));

  }//end main

}//end class Code03
```

Note:
Listing 4

```
/*************************************************
******************
Copyright 2016 R.G.Baldwin
Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
*************************************************
```

```
******************/

import org.json.simple.JSONObject;
import org.json.simple.JSONValue;

class Code04 {

  public static void main(String[] args){

    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("name","Joe");
    jsonObj01.put("age",21);

    String jsonString = jsonObj01.toJSONString();

    JSONObject jsonObj02 =
(JSONObject)JSONValue.parse(jsonString);
    System.out.println(jsonObj02.get("name"));
    System.out.println(jsonObj02.get("age"));

  }//end main

}//end class Code04
```

Note:
Listing 5

```
/*************************************************
******************
Copyright 2016 R.G.Baldwin
Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
*************************************************
```

```
******************/

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

class Code05 {

  public static void main(String[] args){

    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("name","Joe");
    jsonObj01.put("age",21);

    String jsonString = jsonObj01.toJSONString();

    JSONObject jsonObj02 =
(JSONObject)JSONParser.parse(jsonString);
    System.out.println(jsonObj02.get("name"));
    System.out.println(jsonObj02.get("age"));

  }//end main

}//end class Code05
```

Note:
Listing 6

```
/**************************************************
******************
Copyright 2016 R.G.Baldwin
Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
**************************************************
```

```
******************/

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

class Code06 {

  public static void main(String[] args){

    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("name","Joe");
    jsonObj01.put("age",21);

    String jsonString = jsonObj01.toJSONString();

    JSONObject jsonObj02 =
                    (JSONObject)(new
JSONParser().parse(jsonString));
    System.out.println(jsonObj02.get("name"));
    System.out.println(jsonObj02.get("age"));

  }//end main

}//end class Code06
```

Note:
Listing 7

```
/*************************************************
******************
Copyright 2016 R.G.Baldwin
Tested with Java 8, Win 7, and json-simple-
1.1.1.jar.
```

```
**********************************************************
******************/

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

class Code07 {

  public static void main(String[] args) throws
ParseException{

    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("name","Joe");
    jsonObj01.put("age",21);

    String jsonString = jsonObj01.toJSONString();

    JSONObject jsonObj02 =
                    (JSONObject)(new
JSONParser().parse(jsonString));
    System.out.println(jsonObj02.get("name"));
    System.out.println(jsonObj02.get("age"));

  }//end main

}//end class Code07
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0220R: Review
- File: Json0220R.htm
- Published: 06/04/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0225: Encoding JSON Arrays
Learn how to encode JSON array data using the JSONArray class. Also learn how to write an encoded JSON string to an output text file for transport to a different computing environment.

Revised: Thu Jun 02 19:27:08 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of contents

- Miscellaneous

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page explains how to encode JSON array data using the **JSONArray** class. It also shows how to write an encoded JSON string to an output text file for transport to a different programming environment.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

General background information

As you learned in the page titled Json0210: Structure of the json-simple Java Library , the **JSONArray** class extends the standard Java **ArrayList** class. Once you have an object of the **JSONArray** class, you have access to all of the methods defined in and inherited into the **JSONArray** class.

The sample program that follows will use the **JSONArray** class to construct a JSON string containing nested arrays and will then write the string to an output file suitable for transport to a different programming environment.

A later page in this book will read the JSON string from the file and decode it into its component parts.

Discussion and sample code

The program named **Code** *(see Listing 7)* creates a JSON string and writes it to an output file. The **json** string contains an array, which in turn contains two nested arrays.

Beginning of the class and the main method

I will discuss and explain the program in fragments. The first fragment is shown in Listing 1 .

Note: Listing 1 . Beginning of the class and the main method.

```
import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import java.io.PrintWriter;
import java.io.File;
import java.io.IOException;
```

```
class Code{

  public static void main(String[] args){
    //Create a json object. which is a subclass
    // of the Java HashMap class.
    JSONObject hashMapA = new JSONObject();

    //Populate the json object with a key/value
    // pair.
    hashMapA.put("name","Tom");
```

There is nothing new in Listing 1 so that code shouldn't require further explanation. Note however, that the name "Tom" will become significant in a later discussion.

Instantiate and populate a JSONArray object

The code in Listing 2 instantiates and populates a **JSONArray** object.

Note: Listing 2 . Instantiate and populate a JSONArray object.

```
    JSONArray arrayListA = new JSONArray();
    arrayListA.add("2-club");
    arrayListA.add("3-heart");
    arrayListA.add("4-diamond");
    arrayListA.add("5-spade");
```

Previous pages in this book have constructed JSON strings using a subclass of the Java **HashMap** class -- *(the **JSONObject** class)* . It is worth noting that JSON strings constructed in that manner are unordered. However, JSON arrays constructed using the **JSONArray** class, which is a subclass of the **ArrayList** class, are ordered lists.

The program that I will explain in this page is intended to represent the beginning state of a two-person card game where each player receives four cards. The code in Listing 2 constructs a list of the cards that will be dealt to one of the players.

Populate hashMapA with a key/value pair

The code in Listing 3 populates the **JSONObject** object referred to as **hashMapA** with a key/value pair where the key is "cards" and the value is an object of the **JSONArray** class containing a list of strings naming specific playing cards.

Note: Listing 3 . Populate hashMapA.

```
    hashMapA.put("cards",arrayListA);
```

Note that **hashMapA** already contained a key/value pair identifying one of the players in the game named "Tom" *(see Listing 1)*. Thus the code in Listing 1 through Listing 3 can be though of as "dealing" the cards identified in Listing 2 to the player named "Tom".

Create and populate another similar JSON object

Listing 4 creates and populates a second list of playing cards and deals them to the second player in the game whose name is "Joe".

Note: Listing 4 . Create and populate another similar JSON object.

```
JSONObject hashMapB = new JSONObject();
hashMapB.put("name","Joe");

JSONArray arrayListB = new JSONArray();
arrayListB.add("4-heart");
arrayListB.add("5-heart");
arrayListB.add("6-club");
arrayListB.add("7-diamond");
hashMapB.put("cards",arrayListB);
```

Put the players in the game

Now that the players have been created and have received their cards, it is time to put them in the game.

Listing 5 begins by adding the two players and their card arrays to a new object of type **JSONArray** . This results in nested arrays.

Note: Listing 5 . Put the players in the game.

```
JSONArray arrayListC = new JSONArray();
arrayListC.add(hashMapA);
arrayListC.add(hashMapB);
```

```
    JSONObject hashMapC = new JSONObject();
    hashMapC.put("game",arrayListC);
```

Then Listing 5 creates a new **JSONObject** object to represent the game and populates it with a key/value pair where the key is "game" and the value is the array containing the two players and their card arrays.

Write the JSON string to an output file

Listing 6 calls the **writeJSONString** method on the **JSONObject** object to encode the object into a JSON string and write it to an output file named **junk.json** .

Note: Listing 6 . Write the JSON string to an output file.

```
    try{
      PrintWriter out =
          new PrintWriter(new File("junk.json"));
      hashMapC.writeJSONString(out);
      out.flush();
    }catch(IOException ex){
      ex.printStackTrace();
    }//end catch

  }//end main

}//end class Code
```

Note that the output file is simply a text file with the extension **.json** . Thus it can be read by any program that is capable of reading plain text files.

The end of the program

Listing 6 also signals the end of the **main** method and the end of the program.

The contents of the output file

Figure 1 shows a "prettified" version of the contents of the output file.

Note: Figure 1. Prettified version of output file contents.

```
{"game":
  [
    {
      "cards":
      [
        "2-club","3-heart","4-diamond","5-spade"
      ],
      "name":"Tom"
    },
    {
      "cards":
      [
        "4-heart","5-heart","6-club","7-diamond"
      ],
      "name":"Joe"
    }
  ]
}
```

The actual contents of the file do not contain line breaks and indentation as shown in Figure 1 . Those cosmetic features were added manually to Figure 1 to make it easier for you to correlate the output with the code shown earlier. Instead, the actual file consists simply of a string of characters that begins as shown in Figure 2 .

Note: Figure 2. Beginning of output file contents.

```
{"game":[{"cards":["2-club","3-heart","4-
diamond","5-spade"],...
```

Decoded output data

As mentioned earlier, a later page in this book will read the JSON string from the file and decode it into its component parts. Figure 3 shows a preview of what you will see on that page.

Note: Figure 3. Decoded output data.

```
First Player's Name: Tom
First Player's cards
2-club
3-heart
4-diamond
5-spade
```

```
Second Player's Name: Joe
Second Player's cards
4-heart
5-heart
6-club
7-diamond
```

Figure 3 shows the JSON string decoded and formatted into a display that is representative of the intent of the string -- to encapsulate information about the players and their cards in a game of cards.

Run the program

I encourage you to copy the code from Listing 7 . Execute the code and confirm that your output file matches that shown in Figure 2 . Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Complete program listing

Listing 7 provides a complete listing of the program named **Code.java** .

Note: Listing 7 . The program named Code.java.

```
/*********************************************
Copyright: R.G.Baldwin 2016
Revised: 06/01/16

Creates a json string and writes it to an output
file. The json string contains an array, which in
turn contains two arrays.
```

```
Tested with Java 8, Windows 7, and
json-simple-1.1.1.
**********************************************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import java.io.PrintWriter;
import java.io.File;
import java.io.IOException;

class Code{

  public static void main(String[] args){
    //Create a json object. which is a subclass
    // of the Java HashMap class.
    JSONObject hashMapA = new JSONObject();

    //Populate the json object with a key/value
    // pair.
    hashMapA.put("name","Tom");

    //Create and populate a json array, which is
    // a subclass of the Java ArrayList class.
    JSONArray arrayListA = new JSONArray();
    arrayListA.add("2-club");
    arrayListA.add("3-heart");
    arrayListA.add("4-diamond");
    arrayListA.add("5-spade");

    //Populate the json object with a key/value
    // pair where the value is an array.
    hashMapA.put("cards",arrayListA);

    //Create and populate another similar json
    // object.
    JSONObject hashMapB = new JSONObject();
```

```
    hashMapB.put("name","Joe");

    JSONArray arrayListB = new JSONArray();
    arrayListB.add("4-heart");
    arrayListB.add("5-heart");
    arrayListB.add("6-club");
    arrayListB.add("7-diamond");
    hashMapB.put("cards",arrayListB);

    //Create another json array and populate
    // it with the two json objects created
    // earlier.
    JSONArray arrayListC = new JSONArray();
    arrayListC.add(hashMapA);
    arrayListC.add(hashMapB);

    //Create another json object and populate
    // it with a key/value pair where the value
    // is the array from above.
    JSONObject hashMapC = new JSONObject();
    hashMapC.put("game",arrayListC);

    try{
      //Encode the HashMap object into a
      // json String and write it to an output
      // file. Note that it is simply a text file
      // with a different extension.
      PrintWriter out =
          new PrintWriter(new File("junk.json"));
      hashMapC.writeJSONString(out);
      out.flush();
    }catch(IOException ex){
      ex.printStackTrace();
    }//end catch

  }//end main
```

```
}//end class Code
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0225: Encoding JSON Arrays
- File: Json0225.htm
- Published: 06/01/16

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0225R: Review
This page contains review questions and answers for the page titled "Json0225: Encoding JSON Arrays" in the book titled "The json-simple Java Library".

Revised: Sun Jun 05 10:59:02 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of Contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page provides review questions and answers for the page titled Json0225: Encoding JSON Arrays . Once you study that page, you should be able to answer the review questions in this page.

The questions and the answers in this page are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

When a question or an answer provides a link to a figure or a listing, you should open that link in a new window to make it easy to view while reading the question or the answer.

Note:
NOTE:
With respect to the screen output shown on this page, ignore the presence or absence of output similar to the following:

```
Note: Code99.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
```

Questions

Question 1 .

True or False? The **JSONArray** class extends the standard Java **HashMap** class.

Go to answer 1

Question 2

True or False? The **JSONArray** class represents a true Java array. The elements in a **JSONArray** object can be accessed using the square bracket ([]) notation commonly associated with Java arrays.

Go to answer 2

Question 3

True or False? The code in Listing 1 produces the screen output shown in Figure 1 .

Go to answer 3

Question 4

True or False? The code in Listing 2 produces the screen output shown in Figure 3 .

Go to answer 4

Question 5

True or False? The code in Listing 3 produces the screen output shown in Figure 5 .

Go to answer 5

Question 6

True or False? The code in Listing 4 produces the screen output shown in Figure 6 .

Go to answer 6

Question 7

True or False? The code in Listing 5 produces the screen output shown in Figure 8 .

Go to answer 7

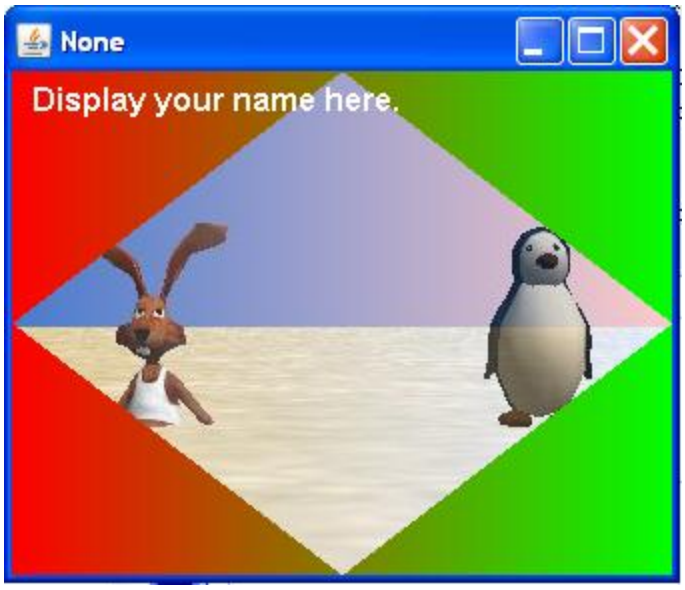
Figure index

Listing index

- Listing 3
- Listing 4
- Listing 5

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 7

True. As a subclass of the **ArrayList** class, an object of the **JSONArray** class allows duplicate elements.

Go back to Question 7

Answer 6

False. The code in Listing 4 produces the screen output shown in Figure 7 .

Go back to Question 6

Answer 5

True. The **JSONArray** class provides two overloaded versions of the method named **toJSONString** . One is an instance method and the other is

a ***static*** method. On a side note, this is also true of the class named **JSONObject** .

Go back to Question 5

Answer 4

False. The code in Listing 2 produces the screen output shown in Figure 4 . Unlike an object of the **JSONObject** class, an object of the **JSONArray** class is an ***ordered*** list. By this we mean that the user has precise control over where in the list each element is inserted. The user can access elements by their integer index *(position in the list)* .

Go back to Question 4

Answer 3

False. The code in Listing 1 produces the screen output with the errors shown in Figure 2 . The **JSONArray** class neither defines nor inherits a method named **put** .

Go back to Question 3

Answer 2

False. The **JSONArray** class extends the **ArrayList** class, which implements the **List** interface. Therefore, an object of the **JSONArray** class is a list and is not an array. For example, the elements in a **JSONArray** object cannot be accessed using the square bracket ([]) notation commonly associated with Java arrays. Instead, the elements are accessed using the **add** and **get** methods declared in the **List** interface and inherited from the **ArrayList** class.

Go back to Question 2

Answer 1

False. The **JSONArray** class extends the standard Java **ArrayList** class.

Go back to Question 1

Figures

This section contains Figures that may be referred to by one or more questions or answers.

Note:
Figure 1

```
[{"Name":"Tom"},{"Age":21},{"Student":true}]
```

Note:
Figure 2

```
Code01.java:22: error: cannot find symbol
array.put(jsonObj01);
^
symbol: method put(JSONObject)
location: variable array of type JSONArray
Code01.java:23: error: cannot find symbol
array.put(jsonObj02);
^
```

```
symbol: method put(JSONObject)
location: variable array of type JSONArray
Code01.java:24: error: cannot find symbol
array.put(jsonObj03);
^
symbol: method put(JSONObject)
location: variable array of type JSONArray
Note: Code01.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
3 errors
Error: Could not find or load main class Code01
Press any key to continue . . .
```

Note:
Figure 3

```
[{"Name":"Tom"},{"Age":21},{"Student":true}]
```

Note:
Figure 4

```
[{"Name":"Tom"},{"Student":true},{"Age":21}]
```

Note:
Figure 5

```
[{"Name":"Tom"},{"Age":21},{"Student":true}]
[{"Name":"Tom"},{"Age":21},{"Student":true}]
```

Note:
Figure 6

```
{"person":[{"Student":true},{"Age":21},
{"Name":"Tom"}]}
```

Note:
Figure 7

```
[{"Student":true},{"Age":21},{"Name":"Tom"}]
```

Note:
Figure 8

```
{"person":[{"Name":"Tom"},{"Age":21},
{"Name":"Tom"}]}
```

Listings

This section contains Listings that may be referred to by one or more questions or answers.

Note:
Listing 1

```
/*********************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*********************************************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;

class Code01{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array = new JSONArray();
    array.put(jsonObj01);
    array.put(jsonObj02);
    array.put(jsonObj03);
    System.out.println(array.toJSONString());

  }//end main
```

```
}//end class Code01
```

Note:
Listing 2

```
/*********************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*********************************************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;

class Code02{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array = new JSONArray();
    array.add(0,jsonObj01);
    array.add(1,jsonObj03);
    array.add(2,jsonObj02);
    System.out.println(array.toJSONString());

  }//end main
```

```
}//end class Code02
```

Note:
Listing 3

```
/*********************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*********************************************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;

class Code03{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array = new JSONArray();
    array.add(jsonObj01);
    array.add(jsonObj02);
    array.add(jsonObj03);

System.out.println(JSONArray.toJSONString(array));
    System.out.println(array.toJSONString());
```

```
  }//end main

}//end class Code03
```

Note:
Listing 4

```
/*************************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*************************************************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;

class Code04{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array = new JSONArray();
    array.add(jsonObj03);
    array.add(jsonObj02);
    array.add(jsonObj01);
```

```
    JSONObject jsonObj04 = new JSONObject();
    jsonObj04.put("person",array);

    System.out.println(array.toJSONString());

  }//end main

}//end class Code04
```

Note:
Listing 5

```
/*******************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*******************************************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;

class Code05{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array = new JSONArray();
```

```
    array.add(jsonObj01);
    array.add(jsonObj02);
    array.add(jsonObj01);

    JSONObject jsonObj04 = new JSONObject();
    jsonObj04.put("person",array);

    System.out.println(jsonObj04.toJSONString());

  }//end main

}//end class Code05
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0225R: Review
- File: Json0225R.htm
- Published: 06/05/16

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0230: Decoding JSON Arrays
Learn how to read a JSON string containing nested array data from a file, decode it, and display its component parts.

Revised: Thu Jun 02 19:28:45 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page explains how to read a JSON string containing nested array data from a file, decode it, and display its component parts.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

General background information

You learned how to encode JSON data containing nested arrays and how to write the encoded JSON string to an output file in the page titled Json0225: Encoding JSON Arrays . The program that I will discuss and explain in the next section will read that file from the disk, decode it, and display its component parts.

Discussion and sample code

The program named **Code** *(see Listing 7)* reads and parses an input file named **junk.json** , The file contains a JSON string with nested array data.

Read the file containing the JSON string

I will discuss and explain this program in fragments. The first fragment is shown in Listing 1 .

Note: Listing 1 . Read the file containing the JSON string.

```
class Code{

  public static void main(String[] args){
    //Instantiate a JSONObject object, which is a
subclass of the
    // Java HashMap class.
    JSONObject jsonMap = null;
    try{
      //Read json string from a file and parse it
into a HashMap.
      jsonMap =
          (JSONObject)(JSONValue.parse(new
FileReader("junk.json")));
    }catch(IOException ex){
```

```
      ex.printStackTrace();
    }//end catch
```

The code in Listing 1 shows the beginning of the class and the beginning of the **main** method. This code reads the JSON string from the input file named **junk.json** and parses the data into an object of type **JSONObject** .

There is nothing new in Listing 1 so the code shouldn't need further explanation. When the code in Listing 1 has finished executing, all of the information from the JSON string in the file is encapsulated in the object of type **JSONObject** referred to as **jsonMap** .

Display the JSON string

The code in Listing 2 displays the raw JSON string followed by a blank line.

Note: Listing 2 . Display the JSON string.

```
    System.out.println("json string: " + jsonMap);
    System.out.println();//blank line
```

This code produces the screen output shown by the screen shot in Figure 1 .

Note: Figure 1. Screen shot of raw JSON string.

```
json string: {"game":[{"cards":["2-club","3-
heart","4-diamond","5-spade"],"name"
:"Tom"},{"cards":["4-heart","5-heart","6-club","7-
diamond"],"name":"Joe"}]}
```

Note that the line break in Figure 1 was inserted by the operating system while displaying the string in the command-line window. The line break does not exist in the data in the file.

Access the game array

Listing 3 calls the **get** method inherited from the **HashMap** class to get the game array into an object of type **JSONArray** , which is a subclass of the **ArrayList** class.

Note: Listing 3 . Access the game array.

```
    JSONArray gameArrayList =
(JSONArray)jsonMap.get("game");
```

Get information about the first player

Recall that a **JSONArray** object is an ordered list as a subclass of the **ArrayList** class. The **get** method inherited from the **ArrayList** class can be used to access elements in the list on the basis of a zero-based index. At this point, information about the first player is stored in the list as an object of

type **JSONObject** . *(It is actually stored as type **Object** and must be downcast to type **JSONObject** in order to do much with it.)*

Listing 4 begins by accessing the element at an index value of zero, which is the object containing information about the first player.

Note: Listing 4 . Get information about the first player.

```
    JSONObject firstPlayerMap =
(JSONObject)gameArrayList.get(0);
    System.out.println("First Player's Name: " +

firstPlayerMap.get("name"));
```

Then Listing 4 uses the **get** method inherited from the **HashMap** class to get and display the value associated with the key "name". This produces the screen output shown in Figure 2 .

Note: Figure 2. Display first player's name.

```
First Player's Name: Tom
```

Display first player's cards

Listing 5 begins by using the **get** method of the **HashMap** class to access the **JSONArray** object that is the value for the key "cards".

Note: Listing 5 . Display first player's cards.

```
    JSONArray firstPlayerCardsList =

(JSONArray)firstPlayerMap.get("cards");

    System.out.println("First Player's cards");

    Iterator<String> iterator =
firstPlayerCardsList.iterator();
    while (iterator.hasNext()) {
      System.out.println(iterator.next());
    }//end while loop
```

Then Listing 5 uses an iterator to iterate through the **JSONArray** object and to display each of the cards in the array, *(which is actually a list at this point)* . This code produces the output shown in Figure 3 .

Note: Figure 3. The first player's cards.

```
First Player's cards
2-club
3-heart
4-diamond
5-spade
```

Get and display information about the second player

Listing 6 uses similar code to get and display information about the second player.

Note: Listing 6 . Get and display information about the second player.

```
    System.out.println();
    JSONObject secondPlayerMap =
(JSONObject)gameArrayList.get(1);
    System.out.println("Second Player's Name: " +

secondPlayerMap.get("name"));

    JSONArray secondPlayerCardsList =

(JSONArray)secondPlayerMap.get("cards");
    System.out.println("Second Player's cards");
    iterator = secondPlayerCardsList.iterator();
    while (iterator.hasNext()) {
      System.out.println(iterator.next());
    }//end while loop

  }//end main

}//end class Code
```

This code produces the screen output shown in Figure 4 .

Note: Figure 4. Information about the second player.

```
Second Player's Name: Joe
Second Player's cards
4-heart
5-heart
6-club
7-diamond
```

The end of the program

Listing 6 signals the end of the **main** method and the end of the program.

Run the program

Click here to download a zip file containing a JSON data file named **junk.json** that can be used to experiment with this program.

I encourage you to copy the code from Listing 7 . Execute the code and confirm that you get the same results as those shown in Figure 1 through Figure 4 . Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Complete program listing

A complete listing of the program named **Code** is provided in Listing 7 .

Note: Listing 7 . The program named Code.java.

```
/**********************************************************
********************
```

```
Copyright: R.G.Baldwin 2016
Revised: 06/01/16

Reads a json string from a file, parses, and
displays some of its parts.

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
**********************************************
******************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.JSONValue;
import java.io.IOException;
import java.io.FileReader;
import java.util.*;

class Code{

  public static void main(String[] args){
    //Instantiate a JSONObject object, which is a
subclass of the
    // Java HashMap class.
    JSONObject jsonMap = null;
    try{
      //Read json string from a file and parse it
into a HashMap.
      jsonMap =
          (JSONObject)(JSONValue.parse(new
FileReader("junk.json")));
    }catch(IOException ex){
      ex.printStackTrace();
    }//end catch

    //Get and display the json string
    System.out.println("json string: " + jsonMap);
```

```
    System.out.println();//blank line

    //Get the game array into an ArrayList object.
    JSONArray gameArrayList =
(JSONArray)jsonMap.get("game");

    //Get and display info about the first player.
    JSONObject firstPlayerMap =
(JSONObject)gameArrayList.get(0);
    System.out.println("First Player's Name: " +

firstPlayerMap.get("name"));

    JSONArray firstPlayerCardsList =

(JSONArray)firstPlayerMap.get("cards");
    System.out.println("First Player's cards");
    Iterator<String> iterator =
firstPlayerCardsList.iterator();
    while (iterator.hasNext()) {
      System.out.println(iterator.next());
    }//end while loop

    //Get and display info about the second
player.
    System.out.println();
    JSONObject secondPlayerMap =
(JSONObject)gameArrayList.get(1);
    System.out.println("Second Player's Name: " +

secondPlayerMap.get("name"));

    JSONArray secondPlayerCardsList =

(JSONArray)secondPlayerMap.get("cards");
    System.out.println("Second Player's cards");
    iterator = secondPlayerCardsList.iterator();
```

```
    while (iterator.hasNext()) {
      System.out.println(iterator.next());
    }//end while loop

  }//end main

}//end class Code
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0230: Decoding JSON Arrays
- File: Json0230.htm
- Published: 06/01/16

nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0230R: Review
This page contains review questions and answers for the page titled "Json0230: Decoding JSON Arrays" in the book titled "The json-simple Java Library".

Revised: Tue Jun 07 10:33:04 CDT 2016

This page is included in the following Books:

- *The json-simple Java Library* .
- *INEW2338 - Advanced Java Programming*
- *Object-Oriented Programming (OOP) with Java*

Table of Contents

Preface

This is a page from the book titled **The json-simple Java Library** . The book explains how to use the **json-simple** Java library to generate, transform, and query JSON text. This page provides review questions and answers for the page titled Json0230: Decoding JSON Arrays . Once you study that page, you should be able to answer the review questions in this page.

The questions and the answers in this page are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

When a question or an answer provides a link to a figure or a listing, you should open that link in a new window to make it easy to view while reading the question or the answer.

Note:
NOTE:
With respect to the screen output shown on this page, ignore the presence or absence of output similar to the following:

```
Note: Code99.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
```

Questions

Question 1 .

True or False? The code in Listing 1 produces the screen output shown in Figure 1 .

Go to answer 1

Question 2

True or False? The code in Listing 2 produces the screen output shown in Figure 3 .

Go to answer 2

Question 3

True or False? The code in Listing 3 produces the screen output shown in Figure 5 .

Go to answer 3

Question 4

True or False? The code in Listing 4 produces the screen output shown in Figure 7 .

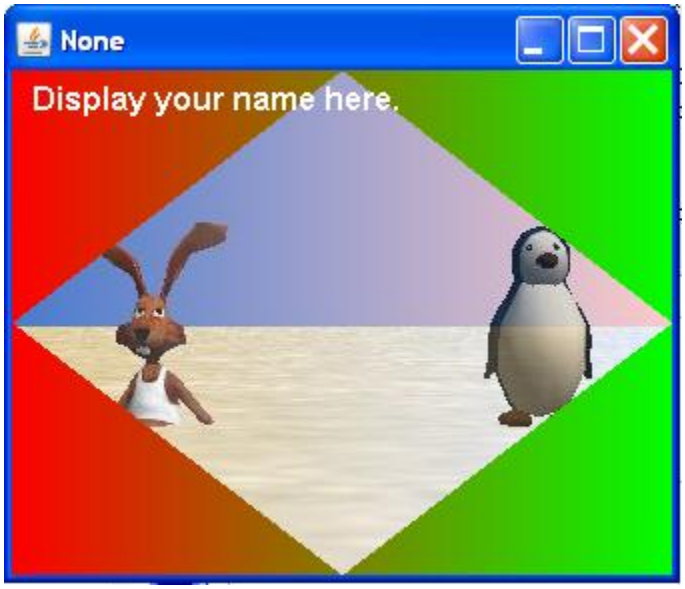
Go to answer 4

Figure index

Listing index

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 4

True.

Go back to Question 4

Answer 3

False. The code in Listing 3 produces the screen output with the error shown in Figure 6 .

Go back to Question 3

Answer 2

False. The code in Listing 2 produces the screen output with the error shown in Figure 4 .

Go back to Question 2

Answer 1

False. The code in Listing 1 produces the screen output with the error shown in Figure 2 .

Go back to Question 1

Figures

This section contains Figures that may be referred to by one or more questions or answers.

Note:
Figure 1

```
Tom
```

Note:
Figure 2

```
Code01.java:36: error: incompatible types: Object
cannot be converted to String
    String temp02 = temp01.get("Name");
                                    ^
Note: Code01.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
```

```
1 error
Error: Could not find or load main class Code01
```

Note:
Figure 3

```
Tom
21
```

Note:
Figure 4

```
Code02.java:40: error: incompatible types: String
cannot be converted to Long
    Long temp04 = (String)(temp03.get("Age"));
                  ^
Note: Code02.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
Error: Could not find or load main class Code02
```

Note:
Figure 5

```
Tom
```

```
21
true
```

Note:
Figure 6

```
Code03.java:44: error: incompatible types: String
cannot be converted to Boolean

    Boolean temp06 = (String)
(temp05.get("Student"));
                     ^
Note: Code03.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
Error: Could not find or load main class Code03
```

Note:
Figure 7

```
Tom
21
true
```

Listings

This section contains Listings that may be referred to by one or more questions or answers.

Note:
Listing 1

```
/*********************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*********************************************
******************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.JSONValue;

class Code01{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array01 = new JSONArray();
    array01.add(jsonObj01);
    array01.add(jsonObj02);
    array01.add(jsonObj03);

    JSONObject jsonObj04 = new JSONObject();
```

```
    jsonObj04.put("person",array01);

    String jsonString = jsonObj04.toJSONString();

    JSONObject jsonObj05 = (JSONObject)
(JSONValue.parse(jsonString));
    JSONArray array02 = (JSONArray)
(jsonObj05.get("person"));

    JSONObject temp01 = (JSONObject)
(array02.get(0));
    String temp02 = temp01.get("Name");
    System.out.println(temp02);

  }//end main

}//end class Code01
```

Note:

Listing 2

```
/*************************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*************************************************
*****************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.JSONValue;
```

```
class Code02{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array01 = new JSONArray();
    array01.add(jsonObj01);
    array01.add(jsonObj02);
    array01.add(jsonObj03);

    JSONObject jsonObj04 = new JSONObject();
    jsonObj04.put("person",array01);

    String jsonString = jsonObj04.toJSONString();

    JSONObject jsonObj05 = (JSONObject)
(JSONValue.parse(jsonString));
    JSONArray array02 = (JSONArray)
(jsonObj05.get("person"));

    JSONObject temp01 = (JSONObject)
(array02.get(0));
    String temp02 = (String)(temp01.get("Name"));
    System.out.println(temp02);

    JSONObject temp03 = (JSONObject)
(array02.get(1));
    Long temp04 = (String)(temp03.get("Age"));
    System.out.println(temp04);

  }//end main
```

```
}//end class Code02
```

Note:
Listing 3

```
/*********************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
*********************************************
******************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.JSONValue;

class Code03{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array01 = new JSONArray();
    array01.add(jsonObj01);
    array01.add(jsonObj02);
    array01.add(jsonObj03);
```

```
    JSONObject jsonObj04 = new JSONObject();
    jsonObj04.put("person",array01);

    String jsonString = jsonObj04.toJSONString();

    JSONObject jsonObj05 = (JSONObject)
(JSONValue.parse(jsonString));
    JSONArray array02 = (JSONArray)
(jsonObj05.get("person"));

    JSONObject temp01 = (JSONObject)
(array02.get(0));
    String temp02 = (String)(temp01.get("Name"));
    System.out.println(temp02);

    JSONObject temp03 = (JSONObject)
(array02.get(1));
    Long temp04 = (Long)(temp03.get("Age"));
    System.out.println(temp04);

    JSONObject temp05 = (JSONObject)
(array02.get(2));
    Boolean temp06 = (String)
(temp05.get("Student"));
    System.out.println(temp06);

  }//end main

}//end class Code03
```

Note:
Listing 4

```
/**********************************************
Copyright: R.G.Baldwin 2016

Tested with Java 8, Windows 7, and
json-simple-1.1.1.
**********************************************
******************/

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.JSONValue;

class Code04{

  public static void main(String[] args){
    JSONObject jsonObj01 = new JSONObject();
    jsonObj01.put("Name","Tom");
    JSONObject jsonObj02 = new JSONObject();
    jsonObj02.put("Age",21);
    JSONObject jsonObj03 = new JSONObject();
    jsonObj03.put("Student",true);

    JSONArray array01 = new JSONArray();
    array01.add(jsonObj01);
    array01.add(jsonObj02);
    array01.add(jsonObj03);

    JSONObject jsonObj04 = new JSONObject();
    jsonObj04.put("person",array01);

    String jsonString = jsonObj04.toJSONString();

    JSONObject jsonObj05 = (JSONObject)
(JSONValue.parse(jsonString));
    JSONArray array02 = (JSONArray)
(jsonObj05.get("person"));
```

```
    JSONObject temp01 = (JSONObject)
(array02.get(0));
    String temp02 = (String)(temp01.get("Name"));
    System.out.println(temp02);

    JSONObject temp03 = (JSONObject)
(array02.get(1));
    Long temp04 = (Long)(temp03.get("Age"));
    System.out.println(temp04);

    JSONObject temp05 = (JSONObject)
(array02.get(2));
    Boolean temp06 = (Boolean)
(temp05.get("Student"));
    System.out.println(temp06);

  }//end main

}//end class Code04
```

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0230R: Review
- File: Json0230R.htm
- Published: 06/07/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-